Isaac Ocegueda

2/23/2025

CS 470 Final Reflection

YouTube URL: https://youtu.be/pt_mDP4GLJY

Experiences and Strengths

In this course, I learned how to use tools to move a full stack application to the cloud. These tools included Docker for containerization, Docker Compose for orchestration, and AWS for the database, gateway, compute service, and security required to run the application. The AWS services I learned to use were DynamoDB, Lambda, API Gateway, and IAM. These skills help me become more marketable in my aspiring field of software development since cloud computing and serverless applications are starting to take over the market due to the many benefits they offer over traditional data servers. Being a developer experienced with these tools will be an advantage over developers who have no knowledge or are still learning.

My strengths as a software developer include the ability to analyze a problem and use the tools available to solve the problem. My ability to learn new skills and quickly adapt is another strength that is very useful in the ever-evolving software development field. A third strength is communication, including my ability to break down technically complicated topics and explain them in terms that are easy to follow and understand.

The types of roles I am prepared to assume include forming part of a development team tasked with architecting and deploying new application or systems. Throughout my courses, I learned about the Scrum process and how a team of developers work together in sprints to complete a project. I am prepared to form a part of a Scrum team and work with other developers to plan, code, and test throughout the project. In the example of this course, I would be prepared to work on the database, functions, or routing of the gateway simultaneously with other developers to quickly create and deploy the application.

Planning for Growth

Growing the application involves scaling and error handling. For scaling, AWS offers

automated and instant scaling services for the database through their S3 service, and Lambda functions revolve around user activity by only activating when they are called on. This automates the application's capacity and usage as determined by the users' demand. For error handling, AWS offers a service called Step Functions that allows the creation of a serverless workflow dedicated to function error handling. This allows error handling to be separated from the business logic of Lambda functions.

AWS also has a service to help predict costs. Cost Explorer analyzes the current and last 13 months of data usage and uses that info to predict the next 12 months' usage and costs. With this forecast, I can set alarms and budgets. This will help me ensure that costs are not too high and help me ramp down capacity if I start going over the budget limit. Of course, forecasts are not entirely accurate, and using a serverless framework can be very unpredictable compared to containers. Containers are better for applications with steady resource use.

Expanding requires a lot of planning and calculating. A pro of expanding is the ability to grow the app and create more revenue from its usage. Expanding could also mean more features that attract new users. However, more usage does mean a higher resource cost for both data storage and activity usage. If these costs were to ramp up faster than the number of users, it would mean a loss of revenue and possibly the need to scale down the application.

Elasticity is an important principle that helps deal with app expansion. As the app adds new features and gains new users, the app's resources must also match its growth. In a traditional data server, this growth in resources is seen in a step model, where resources are added periodically with a set new ceiling after each upgrade. However, this process can take a significant amount of time to accomplish and leave users with an app that is unresponsive until those upgrades are made. On the other hand, AWS uses a pay-for-service approach that

automatically scales up or down depending on usage. This means high elasticity that can mold to actual usage and ensure resources are available for the app to function properly. However, as discussed previously, pay-for-service is unpredictable and could lead to unexpectedly high costs. When planning for future growth, resource costs are usually the deciding factor in choosing either a framework with a highly adaptable elasticity or a low but predictable elasticity.